|  |  |
| --- | --- |
| ***CS2NN16 MLP Lab 1 Report Sheet 2017/18*** | |
| **Student Number: 250202184** | **Date :27/10/2017** |

|  |  |
| --- | --- |
| **Introduction** | **Mark / 3** |
| This is lab 1 of Neural Networks module. Specifically in this session, 3 java files are used, one for datasets, and one each for a single layer perceptron network with linear and sigmodal activation. These networks will attempt to learn the logic problems, AND, OR and XOR, displaying the results in the console. Task of this lab is to complete some of the methods in those files, so the program can learn and output results successfully. | |

|  |  |
| --- | --- |
| **Output of Untrained LinearLayerNetwork network** | **Mark / 2** |
| Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 0.200 0.300 0.400 0 0 0  0 1 0 1 1 0.500 0.400 0.600 1 0 1  1 0 0 1 1 0.700 0.800 0.500 1 1 1  1 1 1 1 0 1.000 0.900 0.700 1 1 1  Over Set : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Weights  Epoch 1 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Epoch 2 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Epoch 3 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Epoch 4 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Epoch 5 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Epoch 6 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Epoch 7 : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 0.200 0.300 0.400 0 0 0  0 1 0 1 1 0.500 0.400 0.600 1 0 1  1 0 0 1 1 0.700 0.800 0.500 1 1 1  1 1 1 1 0 1.000 0.900 0.700 1 1 1  Over Set : SSE 0.195 0.125 0.265 : %Correct 50 75 75  Weights | |

|  |  |  |
| --- | --- | --- |
| **LinearLayerNetwork** | | |
| **getWeights** | **Code Mark / 3** | **Comment / 2** |
| /\*\*  \* return the weights in the neuron as a string  \* **@return** the string  \*/  **public** String getWeights() {  String s = ""; // set string to empty    //For loop to through all weights  **for** (**int** i=0; i<weights.size(); i++) {  //Format each weight to string (with 3 digits after decimal),  //And add it to final string  s+=String.*format*("%.3f", weights.get(i)) + " ";  }    **return** s; // return the result  } | | |

|  |  |
| --- | --- |
| **Weights of Untrained LinearLayerNetwork network** | **Mark / 1** |
| Weights 0.200 0.500 0.300 0.300 0.500 0.100 0.400 0.100 0.200 | |

|  |  |  |
| --- | --- | --- |
| **findDeltas** | **Code / 3** | **Comments / 2** |
| /\*\*  \* find deltas using the errors passed  \* **@param** errors  \*/  **protected** **void** findDeltas(ArrayList<Double> errors) {  //Go through all errors  **for** (**int** i=0; i<errors.size(); i++) {  //Get all deltas  deltas.set(i, errors.get(i));  }  } | | |

|  |  |  |
| --- | --- | --- |
| **changeAllWeights** | **Code / 5** | **Comments / 3** |
| /\*\*  \* change all the weights in the layer of neurons  \* **@param** ins array list of the inputs to the neuron layer  \* **@param** learnRate learning rate: change is learning rate \* input \* delta  \* **@param** momentum momentum constant : change is also momentun \* change in weight last time  \*/  **protected** **void** changeTheWeights(ArrayList<Double> ins, **double** learnRate, **double** momentum) {    //Double for loop - 1st loop goes through each neuron, 2nd loop goes through each weight  //Counters: nct - neurals counter, ict - inputs counter, wct - weights counter    //Go through each neuron  **for** (**int** nct=0; nct<numNeurons; nct++) {  **double** theIn;  //Go through each input (+1 because of bias input)  **for** (**int** ict = 0; ict < ins.size() + 1; ict++) {  **int** wct = ict + (nct\*3); //Calculate weights counter  //If bias input, set it to 1.0  **if** (ict == 0) theIn = 1.0; **else** theIn = ins.get(ict-1);  //Calculate changeInWeights  changeInWeights.set(wct, theIn\*deltas.get(nct)\*learnRate +  changeInWeights.get(wct) \* momentum);  //Calculate weights  weights.set(wct, weights.get(wct ) +  changeInWeights.get(wct));  }  }  } | | |

|  |  |
| --- | --- |
| **Modify main so learning rate is 0.1 and 0.3. Paste below how the SSE changes during training, the data set after training and the weights** |  |
| **Mark /3** |
| Learning rate: 0.1, momentum 0.3, 10 epochs  Weights 0.20000 0.50000 0.30000 0.30000 0.50000 0.10000 0.40000 0.10000 0.20000  Epoch 1 : SSE 0.185 0.130 0.338 : %Correct 75 75 50  Epoch 2 : SSE 0.121 0.103 0.332 : %Correct 100 100 25  Epoch 3 : SSE 0.107 0.096 0.331 : %Correct 100 100 25  Epoch 4 : SSE 0.100 0.092 0.329 : %Correct 100 100 25  Epoch 5 : SSE 0.096 0.089 0.327 : %Correct 100 100 25  Epoch 6 : SSE 0.093 0.086 0.326 : %Correct 100 100 25  Epoch 7 : SSE 0.090 0.085 0.324 : %Correct 100 100 25  Epoch 8 : SSE 0.088 0.084 0.323 : %Correct 100 100 25  Epoch 9 : SSE 0.087 0.083 0.322 : %Correct 100 100 25  Epoch 10 : SSE 0.085 0.082 0.322 : %Correct 100 100 25  Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 -0.169 0.332 0.474 -0 0 0  0 1 0 1 1 0.246 0.728 0.505 0 1 1  1 0 0 1 1 0.307 0.793 0.452 0 1 0  1 1 1 1 0 0.722 1.189 0.484 1 1 0  Over Set : SSE 0.065 0.066 0.251 : %Correct 100 100 75  Weights -0.16924 0.47649 0.41478 0.33211 0.46119 0.39605 0.47393 -0.02181 0.03148 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code for SigmoidLayerNetwork – mark scheme** | | | | |
| **SigmoidLayerNetwork CalcOutputs** | | **Code / 3** | **Comments / 2** | |
| /\*\*  \* calcOutputs of neuron  \* **@param** nInputs arraylist with the neuron inputs  \* Calcs Sigmoid(weighted sum) where weighted sum being weight(0) + inputs(0..n) \* weights(1..n+1)  \*/  **protected** **void** calcOutputs(ArrayList<Double> nInputs) {  **super**.calcOutputs(nInputs); //Calculate outputs with inherited function  //Go through each output  **for** (**int** i=0; i<outputs.size(); i++) {  outputs.set(i, 1.0/(1.0 + Math.*exp*(-outputs.get(i)))); // calculate sigmoid  }  } | | | | |
| **SigmoidLayerNetwork findDeltas** | **Code / 3** | | | **Comments / 2** |
| /\*\*  \* find deltas, being errors (which are passed to function) \* outputs \* (1 - outputs)  \* **@param** errors  \*/  **protected** **void** findDeltas(ArrayList<Double> errors) {  //Go through all errors  **for** (**int** i=0; i<errors.size(); i++) {  //Calculate each delta  deltas.set(i, errors.get(i)\*(1-outputs.get(i))\*outputs.get(i));  }  } | | | | |

|  |  |
| --- | --- |
| **Program output with default weights, a learning rate of 0.15 and momentum of 0.4 : show state *before*, *during* and then *after* 1000 epochs of training** | **Mark /3** |
| Weights 0.20000 0.50000 0.30000 0.30000 0.50000 0.10000 0.40000 0.10000 0.20000  Epoch 1 : SSE 0.185 0.130 0.338 : %Correct 75 75 50  Epoch 2 : SSE 0.121 0.103 0.332 : %Correct 100 100 25  Epoch 3 : SSE 0.107 0.096 0.331 : %Correct 100 100 25  Epoch 4 : SSE 0.100 0.092 0.329 : %Correct 100 100 25  Epoch 5 : SSE 0.096 0.089 0.327 : %Correct 100 100 25  Epoch 6 : SSE 0.093 0.086 0.326 : %Correct 100 100 25  Epoch 7 : SSE 0.090 0.085 0.324 : %Correct 100 100 25  Epoch 8 : SSE 0.088 0.084 0.323 : %Correct 100 100 25  Epoch 9 : SSE 0.087 0.083 0.322 : %Correct 100 100 25  Epoch 10 : SSE 0.085 0.082 0.322 : %Correct 100 100 25  Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 -0.169 0.332 0.474 -0 0 0  0 1 0 1 1 0.246 0.728 0.505 0 1 1  1 0 0 1 1 0.307 0.793 0.452 0 1 0  1 1 1 1 0 0.722 1.189 0.484 1 1 0  Over Set : SSE 0.065 0.066 0.251 : %Correct 100 100 75  Weights -0.16924 0.47649 0.41478 0.33211 0.46119 0.39605 0.47393 -0.02181 0.03148 | |

|  |  |
| --- | --- |
| **Discussion (on code and results)** | **Mark / 5** |
| When creating SigmoidLayerNetwork methods, a lot of code can be reused from LinearLayerNetwork class, as most of the maths and variables are pretty much the same. Main difference is that deltas are calculated with a different formula, and calculated outputs also need to be calculated with sigmoid.  Because Java has super() method, a lot of time can be saved by not rewriting the same code.  changeTheWeights() method can be fairly difficult to implement, as there are 3 different counters to be used. But by reusing some of the code provided in the lecture notes, it was fairly straightforward to implement.  Looking at results, it’s clear that increasing the count of epochs provides more reliable results. | |
| **Conclusion** | **Mark / 5** |
| Implementing neural networks in Java is not that complex, as long as person understands the Maths behind it and Java language itself.  It also seems like sigmoidLayerNetwork is more successful at learning the problem, though it might look a small bit more complex to understand.  Overall, the lab session was useful for myself personally, as a good practice to implement mathematical formulas into an actual programming language code. | |

|  |  |  |
| --- | --- | --- |
| **Self Evaluation (answer yes/no/maybe)** | **Your View** | **Markers View** |
| My code works fully | **YES** |  |
| My code is clear and concise | **YES** |  |
| Each function has good comments explaining what it does and its arguments | **YES** |  |
| The code implementing the functions are well explained | **YES** |  |
| I understand the code in these classes | **YES** |  |

|  |
| --- |
| **Write below any issues you have or any questions you would like answered** |
|  |

|  |  |
| --- | --- |
| **Markers Comments** | **Total Mark / 50** |
|  | |